TR-0826

Incorporating Top-Down Information into Bottom-Up Hypothetical Reasoning

by
Y. Ohta & K. Inoue

December, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Incorporating Top-Down Information into Bottom-Up Hypothetical Reasoning

Yoshihiko Ohta* **Katsumi Inoue**
ICOT Research Center,
Institute for New Generation Computer Technology,
Mita Kokusai Bldg., 21F, 1-4-28 Mita, Minato-ku, Tokyo 108, Japan.

Abstract

A bottom-up hypothetical reasoner with the assumption-based truth maintenance system (ATMS) has some advantages such as avoiding repeated proofs. However, it may prove subgoals unrelated to proofs of a given goal. In order to simulate top-down reasoning on bottom-up reasoners, we can apply the upside-down meta-interpretation method, which is similar to Magic Set and Alexander methods, by transforming a set of Horn clauses into a program incorporating goal information. Unfortunately, it does not achieve speedups for bottom-up hypothetical reasoning because checking consistencies of solutions by negative clauses should be globally evaluated. This paper presents a new method to reduce the consistency checking cost for bottom-up hypothetical reasoning based on the upside-down meta-interpretation. In the transformation algorithm, logical dependencies between a goal and negative clauses are analyzed to find irrelevant negative clauses, so that bottom-up hypothetical reasoning based on the upside down meta-interpretation can restrict consistency checking of negative clauses to those relevant clauses. The transformed program has been evaluated with a logic circuit design problem.

Keywords: Hypothetical reasoning, Default reasoning, ATMS, Upside-down meta-interpretation, Program transformation.

*Current Address: Department of Information Engineering, University of Industrial Technology, 4-1-1 Hashimotodai, Sagamihara, Kanagawa 229, Japan.

1 Introduction

Hypothetical reasoning [9] is an inference technique for proving the given goal from axioms together with a set of hypotheses that do not contradict with the axioms. It is closely related to abductive reasoning and default reasoning. Theorist [17, 18] is a typical hypothetical reasoner based on *top-down* reasoning.

On the other hand, *bottom-up* hypothetical reasoners have been proposed in [6, 12, 15]. These reasoners are constructed by combining a RETE-based inference engine [7] with the assumption-based truth maintenance system (ATMS) [4]. Recently, we have implemented a bottom-up hypothetical reasoner, which consists of the ATMS and the model generation theorem prover called MGTP [8]. In this paper, we see the MGTP as a bottom-up inference engine.

As compared with top-down reasoning, bottom-up reasoning has the advantage of avoiding duplicate proofs of repeated subgoals. Bottom-up reasoning, however, has the disadvantage of proving unnecessary subgoals that are unrelated to proofs of a given goal.

In order to avoid the disadvantage of bottom-up reasoning, Magic Set method [1] and Alexander method [20] have been proposed for deductive database systems. Each method transforms a given set of definite clauses into a program that does not derive unnecessary unit clauses in proving a goal. Recently, it is shown that Magic Set and Alexander methods are interpreted as specializations of the *upside-down meta-interpretation* [3].

The upside down meta-interpretation has been extended to abduction by Stickel [22]. However, this framework does not require the consistency of abductive solutions. Since the consistency requirement is crucial for some applications, we would like to include negative clauses in programs for hypothetical reasoning to reject inconsistent solutions. However, we will show that the upside down meta interpretation method does not achieve speedups in general if programs include negative clauses. This is because the consistency of solutions by negative clauses should be globally evaluated.

We thus present a new program transformation method for efficient bottom-up hypothetical reasoning based on the upside-down meta-interpretation. In the transformation method, logical dependencies between a goal and negative clauses are analyzed at an abstracted level to find irrelevant negative clauses. As a result of this analysis, called an *abstracted dependency analysis*, bottom up hypothetical reasoning based on the upside-down meta-interpretation can restrict consistency checking with negative clauses to those relevant clauses. The transformed program has been evaluated with a logic circuit design problem.

In Section 2, our hypothetical reasoning is defined with Reiter's default proofs [19]. In Section 3, the outline of the ATMS is sketched. Section 4 shows a basic algorithm for bottom-up hypothetical reasoning with the MGTP and the ATMS. Section 5 introduces the upside-down meta-interpretation for bottom-up hypothetical reasoning, and illustrates that a simple program transformation method does not achieve speedups. In Section 6, we propose a transformation method with an abstracted dependency analysis, and illustrate that an example program transformed by this method is efficient. Section 7 shows a result of an experiment to evaluate the transformed programs. Section 8 considers related work.

2 Problem Definition

In this section, we define our hypothetical reasoning based on a subset of Reiter's normal default theory [19]. A *normal default theory* (D, W) is given as follows:

- W: a set of Horn clauses.

  A *Horn clause* is represented in an implicational form,

  $$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta \tag{1}$$

  or

  $$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \bot. \tag{2}$$

  Here, α_i $(1 \leq i \leq n; n \geq 0)$ and β are atomic formulas, and $\bot$ designates falsity. All variables in a clause are assumed to be universally quantified at the front of the clause. Each Horn clause has to be *range-restricted*, that is, all variables in the *consequent* β have to appear in the *antecedent* $\alpha_1 \wedge \cdots \wedge \alpha_n$. A Horn clause of the form (2) is called a *negative clause*.

- D: a set of normal defaults.

  A *normal default* is an inference rule,

  $$\frac{\alpha : \beta}{\beta}, \tag{3}$$

  where α, called the *prerequisite* of the normal default, is restricted to a conjunction $\alpha_1 \wedge \cdots \wedge \alpha_n$ of atomic formulas and β, called its *consequent*, is restricted to an atomic formula. All variables in the consequent β have to appear in the prerequisite α. A normal default with free variables is identified with the set of its ground instances.

A *goal* G is a conjunction of atomic formulas. All variables in G are assumed to be existentially quantified.

Let Δ be the set of all ground instances of the normal defaults of D. A *default proof* [19] of G with respect to (D, W) is a sequence $\Delta_0, \cdots, \Delta_k$ of subsets of Δ if and only if

1. $W \cup CONSEQUENTS(\Delta_0) \vdash G$,
2. for $1 \leq i \leq k$, $W \cup CONSEQUENTS(\Delta_i) \vdash PREREQUISITES(\Delta_{i-1})$,
3. $\Delta_k = \emptyset$,
4. $W \cup \bigcup_{i=0}^{k} CONSEQUENTS(\Delta_i)$ is consistent,

where

$$PREREQUISITES(\Delta_{i-1}) \equiv \bigwedge_{(\alpha:\beta/\beta) \in \Delta_{i-1}} \alpha$$

and

$$CONSEQUENTS(\Delta_i) \equiv \{\beta \mid (\alpha : \beta/\beta) \in \Delta_i\}.$$

A ground instance $G\theta$ of the goal G is an *answer* to G predicted by (D, W) if

$$W \cup \bigcup_{i=0}^{k} CONSEQUENTS(\Delta_i) \vdash G\theta,$$

where the sequence $\Delta_0, \cdots, \Delta_k$ is a default proof of G with respect to (D, W). If $G\theta$ is an answer to G predicted by (D, W), θ is an *answer substitution* for G predicted by (D, W). A *support* for an answer $G\theta$ predicted by (D, W) is $\bigcup_{i=0}^{k} CONSEQUENTS(\Delta_i)$, where the sequence $\Delta_0, \cdots, \Delta_k$ is a default proof of $G\theta$ with respect to (D, W). For an answer $G\theta$ predicted by (D, W), the set of minimal elements in all supports for $G\theta$ from (D, W) is called the *minimal supports* for $G\theta$ from (D, W), and written as $MS(G\theta)$. The *solution* to G from (D, W) is the set of all pairs $\langle G\theta, MS(G\theta)\rangle$, where $G\theta$ is an answer to G predicted by (D, W) and $MS(G\theta)$ is the minimal supports for $G\theta$. The task of our hypothetical reasoning is defined as finding the solution to a given goal from a given normal default theory.

3 ATMS

The ATMS [4] is used as one component of our hypothetical reasoner. The following is the outline of the ATMS.

In the ATMS, a ground atomic formula is called a *datum*. For some datum N, Γ_N designates an *assumption*, which means that N is assumed to be true. The ATMS treats both $\bot$ and Γ_N as special data. The ATMS represents each datum N as an *ATMS node* $\langle N, L_N, J_N\rangle$, where N is the datum, L_N is the *label* of N and J_N is the *justifications* of N. A *justification* corresponds to a ground Horn clause and is incrementally input to the ATMS. Each justification is denoted by:

$$N_1, \cdots, N_n \Rightarrow N,$$

where N_i $(1 \leq i \leq n; n \geq 0)$ and N are data. Each datum N_i is called an antecedent of the justification, and the datum N is called the consequent of the justification. In J_N, the ATMS records the set of antecedents of justifications whose consequents correspond to N.

Let H be a set of assumptions. An assumption set $E \subseteq H$ is called an *environment*. When we denote an environment by a set of assumptions, each assumption Γ_N is written as N by omitting the letter Γ. An environment E is called *nogood* if $J \cup E$ derives $\bot$, where J is a set of all the justifications input to the ATMS. In L_N, the ATMS records the *label* of the datum N. The label of N is the set of environments $\{E_1, \cdots, E_j, \cdots, E_m\}$ that satisfies the following four properties [4]:

1. N holds in each E_j (soundness),
2. every environment in which N holds is a superset of some E_j (completeness),
3. each E_j is not nogood (consistency),
4. no E_j is a subset of any other (minimality).

If the label of a datum is not empty, the datum is *believed*; otherwise it is not believed. A basic algorithm to compute labels is as follows. When a justification is incrementally input to the ATMS, the ATMS updates the labels relevant to the justification in the following procedure.

Step 1: Let L be the current label of the consequent N of the justification and L_i be the current label of the i-th antecedent N_i of the justification. Set

$$L' = L \cup \{ x \mid x = \bigcup_{i=1}^{n} F_i, \ where \ F_i \in L_i \}.$$

Step 2: Let L'' be the set obtained by removing nogoods and subsumed environments from L'. Set the new label of N to L''.

Step 3: Finish this updating if L is equal to the new label.

Step 4: If N is $\bot$, then remove all new nogoods from labels of all data other than $\bot$.

Step 5: Update labels of the consequents of the recorded justifications which contain N as their antecedents.

4 Hypothetical Reasoning with MGTP and ATMS

The MGTP [8] is a model generation theorem prover, which generates minimal models of a set of range-restricted function-free clauses. In the following, a set of clauses input to the MGTP is called an *MGTP program*. Each clause in an MGTP program is denoted by:

$$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta_1 \vee \cdots \vee \beta_m,$$

where $\alpha_i (1 \leq i \leq n; n \geq 0)$ and $\beta_j (1 \leq j \leq m; m \geq 0)$ are atomic formulas. The MGTP is written in KL1 [23], and works as a bottom-up reasoner on the distributed-memory multiprocessor Multi-PSI and the parallel inference machine PIM/m. Several kinds of bottom-up hypothetical reasoners have been constructed with the MGTP [10]. One implementation of them is illustrated by Figure 1.

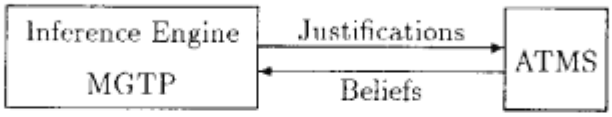

Figure 1: **Bottom-Up Hypothetical Reasoner with MGTP and ATMS**

A normal default theory (D, W) is written as an MGTP program,

$$P \equiv \{ \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \mathbf{assume}(\beta) \mid (\alpha_1 \wedge \cdots \wedge \alpha_n : \beta / \beta) \in D \} \cup W,$$

where **assume** is a metapredicate not appearing anywhere in D and W. In the following, each element in a program P is called a *rule*.

```
procedure R(G, P) :
begin
  B_0 := ∅;
  J_0 := { (⇒ β) | (→ β) ∈ P } ∪ { (Γ_β ⇒ β) | (→ assume(β)) ∈ P };
  s := 0;
  while J_s ≠ ∅ do
    begin
      s := s + 1;
      B_s := UpdateLabels(J_{s-1}, ATMS);
      J_s := GenerateJustifications(B_s, P, B_{s-1})
    end;
  Solution := ∅;
  for each θ such that Gθ ∈ B_s do
    begin
      L_{Gθ} := GetLabel(Gθ, ATMS);
      Solution := Solution ∪ {⟨Gθ, L_{Gθ}⟩}
    end;
  return Solution
end.
```

Figure 2: **Reasoning Algorithm with MGTP and ATMS**

Given a goal G, the reasoning procedure $R(G, P)$ with the MGTP and the ATMS is shown by Figure 2. The reasoning procedure consists of the part for *UpdateLabels–GenerateJustifications* cycles and the part for constructing the solution. The *UpdateLabels–GenerateJustifications* cycles are repeated while a set J_s of new justifications is not empty. The ATMS updates the labels related to a justification set J_{s-1} given by the MGTP. The ATMS returns the set B_s of all the data whose labels are not empty after the ATMS has updated labels with J_{s-1}. The procedure $UpdateLabels(J_{s-1}, ATMS)$ returns a believed data set B_s. The MGTP generates each set J_s of justifications by matching elements of B_s with the antecedent of every rule related to new believed data. The procedure $GenerateJustifications(B_s, P, B_{s-1})$ returns a new justification set J_s. If any element in $(B_s \setminus B_{s-1})$ can match an element of the antecedent of any rule $(\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow X)$ in P and there exists a ground substitution σ for all α_i $(1 \leq i \leq n; n \geq 0)$ such that $\alpha_i\sigma \in B_s$, then J_s is constructed as follows:

- $(\alpha_1\sigma, \cdots, \alpha_n\sigma, \Gamma_{\beta\sigma} \Rightarrow \beta\sigma) \in J_s$ if $X = \mathbf{assume}(\beta)$.
- $(\alpha_1\sigma, \cdots, \alpha_n\sigma \Rightarrow \beta\sigma) \in J_s$ if $X = \beta$.
- $(\alpha_1\sigma, \cdots, \alpha_n\sigma \Rightarrow \bot) \in J_s$ if $X = \bot$.

The procedure $GetLabel(G\theta, ATMS)$ returns the label of $G\theta$, where θ is a ground substitution, and is used in constructing the solution. Note that the label of $G\theta$ corresponds to the minimal

supports for $G\theta$. This hypothetical reasoner can avoid duplicate proofs among different contexts and repeated proofs of subgoals.

5 Upside-Down Meta-Interpretation

Bottom-up reasoning has the disadvantage of proving unnecessarily subgoals that are not related to proofs of the given goal. Here, we introduce a program transformation based on the upside-down meta-interpretation [22] for obtaining speedups of bottom-up reasoning by incorporating goal information.

Let α_i $(1 \leq i \leq n; n \geq 1)$ be an atomic formulas and β be an atomic formula possibly having the metapredicate **assume**. An MGTP program is a set of the form:

$$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta, \tag{4}$$

$$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \bot \tag{5}$$

or

$$\rightarrow \beta. \tag{6}$$

In the following, we call the form (4) a *rule*, the form (5) a *negative clause* and the form (6) a *fact*.

A bottom-up reasoner interprets a rule in such a way that a fact $\beta\sigma$ is derived if facts $\alpha_1\sigma, \cdots, \alpha_n\sigma$ are present for some substitution σ. On the other hand, a top-down reasoner interprets it in such a way that goals $\alpha_1\sigma, \cdots, \alpha_n\sigma$ are derived if a goal $\beta\sigma$ is present, and a fact $\beta\sigma$ is derived if a goal $\beta\sigma$ and facts $\alpha_1\sigma, \cdots, \alpha_n\sigma$ are present. In order to simulate top-down reasoning on bottom up reasoner, we transform a rule into

$$goal(\beta') \rightarrow goal(\alpha_i)$$

for every α_i $(1 \leq i \leq n; n \geq 1)$ and

$$goal(\beta') \wedge \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta,$$

where *goal* is a metapredicate symbol which does not appear in the original program. If $\beta =$ **assume**(X), then $\beta' = X$; otherwise $\beta' = \beta$. Each fact in a program is transformed into itself. Each negative clause in a program is transformed into itself and

$$\rightarrow goal(\alpha_i)$$

for every α_i $(1 \leq i \leq n; n \geq 1)$. This is because every nogood has to be computed to guarantee consistencies of solutions. Let G be a given goal, then $\rightarrow goal(G)$ is added into the transformed program, and the program with $\rightarrow goal(G)$ is evaluated in bottom-up manner. Minimal supports for each generated fact relevant to the solution are maintained by the ATMS.

We can also generate a program which works with a left-to-light strategy. If we have a rule in a program, transform it to the following.

1. $goal(\beta') \rightarrow goal(\alpha_1)$,
   $goal(\beta') \rightarrow cont_{k,1}(\mathbf{V})$.

2. for $1 \leq j \leq n - 1$,
   $cont_{k,j}(\mathbf{V}) \wedge \alpha_j \rightarrow goal(\alpha_{j+1})$,
   $cont_{k,j}(\mathbf{V}) \wedge \alpha_j \rightarrow cont_{k,(j+1)}(\mathbf{V})$.

3. $cont_{k,n}(\mathbf{V}) \wedge \alpha_n \rightarrow \beta$.

Here, we assume that the original rule is named k. These transformed rules work in the following way.

1. If a goal for β' is present, a goal for α_1 is derived, and for $cont_{k,1}(\mathbf{V})$, where $\mathbf{V}$ is a tuple of all the variables appearing in the original rule, is also derived for generating the next subgoal α_2.

2. The following process is repeated for $1 \leq j \leq n - 1$. The next subgoal α_{j+1} and $cont_{k,(j+1)}(\mathbf{V})$ are derived, if $cont_{k,j}(\mathbf{V})$ is present and α_j has been solved. The tuple $\mathbf{V}$ is used for propagating substitutions for some solved variables of α_j to the next subgoal $goal(\alpha_{j+1})$.

3. The fact β is derived if $cont_{k,n}(\mathbf{V})$ is present and α_n has been solved.

Every fact in the original program is also transformed into itself, and every negative clause is transformed as follows.

1. $\rightarrow goal(\alpha_1)$,
   $\rightarrow cont_{k,1}(\mathbf{V})$.

2. for $1 \leq j \leq n - 1$,
   $cont_{k,j}(\mathbf{V}) \wedge \alpha_j \rightarrow goal(\alpha_{j+1})$,
   $cont_{k,j}(\mathbf{V}) \wedge \alpha_j \rightarrow cont_{k,(j+1)}(\mathbf{V})$.

3. $cont_{k,n}(\mathbf{V}) \wedge \alpha_n \rightarrow \bot$.

In this case, first subgoals of negative clauses are evaluated because the reasoner does not know whether those subgoals are independent on consistency checks of solutions or not. When G is given as the goal, $\rightarrow goal(G)$ is added into the transformed program, and the program with $\rightarrow goal(G)$ is evaluated in bottom-up manner. A program generated by this transformation is called a *transformed program with the left-to-right strategy*.

Note that all the transformed rules do not satisfy the range-restricted condition. In order to make every transformed rule range-restricted, the method introducing magic predicates with *adornments* [2] is proposed for Magic Set method. In short, an adornment indicates an input or output mode of each variable in top-down evaluation of each subgoal. We can assume that each predicate appearing in an original program has a unique adornment because we can use a renamed predicate for the predicate if there is a predicate with different adornments. Then, we can apply a similar technique to the upside-down meta-interpretation method. For a consequent

of a transformed rule, an output-mode variable which does not appear in the antecedent is replaced with a new constant (skolem constant). Moreover, for an antecedent of a transformed rule, all output mode variables of each predicate in *goal* and all output-mode variables in *cont* are replaced with anonymous variables.

Example 1 This example is a design problem for simple logic circuits. Assume that the given program P is as follows.

1. $cla(X) \wedge bit(X, B) \wedge word(W) \rightarrow$ **assume**$(adder(c(X, W/B)))$.
   /* If X is a carry lookahead adder and an input is expressed in B bits and a word is expressed in W bits, then we can assume that the combination of X's, whose number of cells is W/B, is an adder for the word as long as it satisfies area constraints. */

2. $inverter(X) \wedge bit(X, B) \wedge word(W) \rightarrow$ **assume**$(cmpl(c(X, W/B)))$.
   /* If X is an inverter and an input is expressed in B bits and a word is expressed in W bits, then we can assume that the combination of X's, whose number of cells is W/B, is a circuit for one's complements as long as it satisfies area constraints. */

3. $adder(X) \wedge cmpl(Y) \rightarrow subtracter(c(X, Y))$.
   /* A combination of an adder X and a circuit Y for one's complements is a subtracter. */

4. $adder(c(X, N)) \wedge cell(X, C) \rightarrow area\_for\_Adder(N * C)$.
   /* If a combination of X's, whose number of cells is N, is an adder and the basic-cell count is N, then the area for the adder is $N * C$. */

5. $cmpl(c(X, N)) \wedge cell(X, C) \rightarrow area\_for\_Cmpl(N * C)$.
   /* If a combination of X's, whose number of cells is N, is a circuit for one's complements and the basic-cell count of X is C, then the area for the one's-complement circuit is $N * C$. */

6. $subtracter(c(c(A, Na), c(C, Nc))) \wedge cell(A, Ca) \wedge cell(C, Cc) \rightarrow area\_for\_Subtracter(Na * Ca + Nc * Cc)$.
   /* If $c(c(A, Na), c(C, Nc))$ is a subtracter and Ca is A's basic-cell count and Cc is C's basic cell count, then the area for the subtracter is $Na * Ca + Nc * Cc$. */

7. $area\_for\_Adder(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.
   /* If an area for any adder is A and an area limit for any circuit is N, then A is less than or equal to N. */

8. $area\_for\_Cmpl(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.
   /* If an area for any circuit for one's complements is A and an area limit for any circuit is N, then A is less than or equal to N. */

9. $area\_for\_Subtracter(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.
   /* If an area for any subtracter is A and an area limit for any circuit is N, then A is less than or equal to N. */

10. $\rightarrow cla(a2h)$. /* The CMOS standard cell $a2h$ is a carry-lookahead adder. */

11. $\rightarrow bit(a2h, 2)$. /* An input to $a2h$ is expressed in 2 bits. */

12. $\rightarrow cell(a2h, 16)$. /* The basic-cell count of $a2h$ is 16. */

13. $\rightarrow cla(a4h)$. /* The CMOS standard cell $a4h$ is a carry-lookahead adder. */

14. $\rightarrow bit(a4h, 4)$. /* An input to $a4h$ is expressed in 4 bits. */

15. $\rightarrow cell(a4h, 50)$. /* The basic-cell count of $a4h$ is 50. */

16. $\rightarrow inverter(v1n)$. /* The CMOS standard cell $v1n$ is an inverter. */

17. $\rightarrow bit(v1n, 1)$. /* An input to $v1n$ is expressed in 1 bit. */

18. $\rightarrow cell(v1n, 1)$. /* The basic-cell count of $v1n$ is 1. */

19. $\rightarrow word(8)$. /* A word is expressed in 8 bits. */

20. $\rightarrow area\_limit(80)$. /* An area limit for every circuit is 80. */

Here, ">" (" is greater than") is a built-in predicate. Assume that the given goal is $adder(X)$, $cmpl(X)$, or $subtracter(X)$. If the given goal is $subtracter(X)$, there are no unnecessary work on the bottom-up hypothetical reasoner. When the given goal is $adder(X)$ or $cmpl(X)$, solving $subtracter(X)$ is unnecessary. Here, let make the simple transformed program P' from the original program P. The simple transformed program P' is:

1. $goal(adder(\_)) \rightarrow goal(cla(sk1))$,
   $goal(adder(\_)) \rightarrow goal(bit(sk1, sk2))$,
   $goal(adder(\_)) \wedge goal(word(sk1))$,
   $goal(adder(\_)) \wedge cla(X) \wedge bit(X, B) \wedge word(W) \rightarrow \mathbf{assume}(adder(c(X, W/B)))$.

2. $goal(cmpl(\_)) \rightarrow goal(inverter(sk1))$,
   $goal(cmpl(\_)) \rightarrow goal(bit(sk1, sk2))$,
   $goal(cmpl(\_)) \wedge goal(word(sk1))$,
   $goal(cmpl(\_)) \wedge inverter(X) \wedge bit(X, B) \wedge word(W) \rightarrow \mathbf{assume}(cmpl(c(X, W/B)))$.

3. $goal(subtracter(\_)) \rightarrow goal(adder(sk1))$,
   $goal(subtracter(\_)) \rightarrow goal(cmpl(sk1))$,
   $goal(subtracter(\_)) \wedge adder(X) \wedge cmpl(Y) \rightarrow subtracter(c(X, Y))$.

4. $goal(area\_for\_Adder(\_)) \rightarrow goal(adder(sk1))$,
   $goal(area\_for\_Adder(\_)) \rightarrow goal(cell(sk1, sk2))$,
   $goal(area\_for\_Adder(\_) \wedge adder(c(X, N)) \wedge cell(X, C) \rightarrow area\_for\_Adder(N * C)$.

5. $goal(area\_for\_Cmpl(\_)) \rightarrow goal(cmpl(sk1))$,
   $goal(area\_for\_Cmpl(\_)) \rightarrow goal(cell(sk1, sk2))$,
   $goal(area\_for\_Cmpl(\_)) \wedge cmpl(c(X, N)) \wedge cell(X, C) \rightarrow area\_for\_Cmpl(N * C)$.

6. $goal(area\_for\_Subtracter(\_)) \rightarrow goal(subtracter(sk1))$,
$goal(area\_for\_Subtracter(\_)) \rightarrow goal(cell(sk1, sk2))$,
$goal(area\_for\ Subtracter(\_)) \wedge subtracter(c(c(A, Na), c(C, Nc))) \wedge cell(A, Ca) \wedge cell(C, Cc) \rightarrow$
$area\_for\_Subtracter(Na * Ca + Nc * Cc)$.

7. $\rightarrow goal(area\_for\_Adder(sk1))$,
$\rightarrow goal(area\_limit(sk1))$,
$area\_for\_Adder(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.

8. $\rightarrow goal(area\_for\_Cmpl(sk1))$,
$area\_for\_Cmpl(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.

9. $\rightarrow goal(area\ for\ Subtracter(sk1))$,
$area\_for\_Subtracter(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.

Moreover, the simple transformed program contains the same facts as the facts in the original program. Let $adder(X)$ be the given goal, then the target program to the bottom-up hypothetical reasoner is the transformed program with $\rightarrow goal(adder(sk1))$ for the given goal. Because of the presence of the negative clause representing constraints on area for subtracters, the fact $goal(area\_for\_Subtracter(sk1))$ is present, and the fact $goal(subtracter(sk1))$ is derived. Therefore, the subgoal $subtracter(X)$ has to be evaluated for consistency checks even if the given goal is $adder(X)$. Then, the computational cost to evaluate the given goal $adder(X)$ with the upside-down meta-interpretation is nearly equal to the cost to evaluate the goal $subtracter(X)$.

6 Abstracted Dependency Analysis

In this section, we propose a static method to find irrelevant negative clauses to evaluation of a given goal. If we can find such irrelevant negative clauses, for every antecedent α_i of each irrelevant clause, we do not need to add $(\rightarrow goal(\alpha_i))$ into the transformed program. We try to find them by analyzing logical dependencies between the goal and each negative clause at an abstracted level as follows. Here, we do not care about any argument in the abstracted dependency analysis. Namely, we analyze dependencies between the predicate symbol of the goal and predicate symbols appearing in negative clauses.

When γ is an atomic formula, we denote by the proposition $\bar{\gamma}$ the predicate symbol of γ. For each negative clause C, the proposition $false_C$ is used as the identifier of C. For every $(\alpha \rightarrow \mathbf{assume}(\beta))$, $\bar{\beta}$ is called an *assumable-predicate symbol*. For any environment E, its *abstracted environment* (denoted by $\bar{E}$) is $\{\ \Gamma_{\bar{\beta}} \mid \Gamma_\beta \in E\}$. The *abstracted justifications* with respect to P is defined as:

$$\begin{aligned}\bar{J} = \ & \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n, \Gamma_{\bar{\beta}} \Rightarrow \beta) \mid (\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \mathbf{assume}(\beta)) \in P\} \\ \cup \ & \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n \Rightarrow \beta) \mid (\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta) \in P\} \\ \cup \ & \{(\bar{\alpha}_1, \cdots, \bar{\alpha}_n \Rightarrow false_C) \mid C = (\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \bot),\ C \in P\}.\end{aligned}$$

Let $\bar{A}$ be the set of propositions appearing in $\bar{J}$. Note that $\bar{A}$ consists of all predicate symbols in P and all $false_C$ for $C \in P$. For each proposition N in $\bar{A}$, we compute the set of abstracted

environments on which N depends. Now, we show an algorithm to compute the set of abstracted environments. This algorithm is obtained with modifications of the label-updating algorithm shown in Section 3. The modified points are as follows.

1. Replace **Step 2** with

   Step 2': Set the new label of N to L'.

2. Remove **Step 4**.

Every proposition in $\bar{A}$ is labeled with the set of abstracted environments obtained by the modified algorithm to the abstracted justifications $\bar{J}$. This label is called the *abstracted label* of the proposition. The system to compute the set of abstracted environments for each proposition is called an *abstracted dependency analyzer*. The reasons why we have to modify the label-updating algorithm are as follows. Firstly, in the abstracted justifications, every $\bot$ is replaced with the proposition $false_C$ for the negative clause C, so that each abstracted label is always consistent. Thus, we do not need **Step 4**. Secondly, each abstracted label may not be minimal because we replace **Step 2** with **Step 2'**. If any abstracted label is minimal, the theorem that we present below may not hold since we require all the dependent assumable-predicate-symbols.

Example 2 An example program is:

$$\begin{aligned} P = \{ \; & \rightarrow p(a), \quad \rightarrow p(b), \quad \rightarrow q(b), \quad q(X) \rightarrow t(X), \\ & p(X) \rightarrow \mathbf{assume}(r(X)), \\ & p(X) \rightarrow \mathbf{assume}(s(X)), \\ & r(a) \rightarrow g, \quad r(X) \wedge s(X) \rightarrow g, \\ & r(X) \wedge s(X) \wedge t(X) \rightarrow \bot \; \}. \end{aligned}$$

Consider the problem defined with the goal g and P. The abstracted label of g is $\{\{r\},\{r,s\}\}$. The abstracted label of the negative clause is $\{\{r,s\}\}$. The abstracted environment $\{r,s\}$ cannot be omitted for g although the set of minimal elements in the abstracted label of g is $\{\{r\}\}$.

Theorem: Let P be a normal default theory, G a goal, $\bar{J}$ the abstracted justifications with respect to P , $L(\bar{G})$ the abstracted label of $\bar{G}$, and $L(false_C)$ the abstracted label of $false_C$ where C is a negative clause in P. If no element in $L(false_C)$ is a subset of any element in $L(\bar{G})$, then the solution to G from P is equivalent to the solution to G from $P \setminus \{C\}$.

Proof: Let C be a negative clause such that $(\alpha \rightarrow \bot) \in P$, where α is a conjunction of atomic formulas. The negation of the sentence is:

1. no element in $L(false_C)$ is a subset of any element in $L(G)$,
2. and the solution to G from P is not equivalent to the solution to G from $P \setminus \{C\}$.

From Sentence 2, an element E_C in the minimal supports $MS(\alpha\sigma_k)$ for an answer $\alpha\sigma_k$ to α predicted by $P \setminus \{C\}$ is a subset of some element E_G in the minimal supports $MS(G\theta_m)$ for some answer $G\theta_m$ to G predicted by $P \setminus \{C\}$. From the definition of abstracted labels, the abstracted environment $\bar{E}_C$ is in the abstracted label $L(false_C)$ of $false_C$ if any environment E_C is in the minimal supports $MS(\alpha\sigma_k)$ for any answer $\alpha\sigma_k$ to α predicted by $P \setminus \{C\}$, and the abstracted environment $\bar{E}_G$ is in the abstracted label $L(\bar{G})$ of $\bar{G}$ if any environment E_G is in

the minimal supports $MS(G\theta_m)$ for any answer $G\theta_m$ to G predicted by $P \setminus \{C\}$. Therefore, if $E_C \subseteq E_G$ for any environments E_C and E_G, then $\bar{E}_C \subseteq \bar{E}_G$ for the abstracted environments $\bar{E}_C$ and E_G. As a consequence, an abstracted environment $\bar{E}_C$ in the abstracted label $L(false_C)$ of $false_C$ is a subset of some abstracted environment $\bar{E}_G$ in the abstracted label $L(\bar{G})$ of $\bar{G}$. This is contrary to Sentence 1. ■

On the basis of the theorem, we can omit consistency checking for a negative clause C if the condition of the theorem is satisfied. By using the theorem repeatedly, we obtain the following.

Corollary: Let P, G, $L(\bar{G})$ be the same as in the theorem. And let

$$\mathcal{C} = \{\ C \mid C \text{ is a negative clause in } P \text{ and no element in } L(false_C) \text{ is a subset of any element in } L(\bar{G})\}.$$

Then, the solution to G from P is equivalent to the solution to G from $P \setminus \mathcal{C}$. ■

Based on the corollary, the transformation algorithm $T(G, P)$ with the abstracted dependency analysis is shown by Figure 3 for the program P and the goal G. In this algorithm, there are two procedure calls to the abstracted dependency analyzer. The meanings of are procedures are as follows:

- $UpdateAbstractedLabels(\bar{J}, ADA)$:
  This procedure computes abstracted labels from abstracted justifications $\bar{J}$ and returns the set ADA of propositions with their abstracted labels.
- $GetAbstractedLabel(\bar{G}, ADA)$:
  This procedure returns the abstracted label of G from the set ADA of propositions with their abstracted labels.

The procedure $T(G, P)$ transforms an original program P into the program in which the top-down information is incorporated and consistency checking is restricted to those negative clauses relevant to the given goal G.

Example 3 Consider again the program P shown in Example 1. Assume that each negative clause in P is named with a number as follows:

- 7: $area\_for\_Adder(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.
- 8: $area\_for\_Cmpl(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.
- 9: $area\_for\_Subtracter(A) \wedge area\_limit(N) \wedge (A > N) \rightarrow \bot$.

The abstracted justifications $\bar{J}$ with respect to the program P are:

1. $(cla, bit, word, \Gamma_{adder} \Rightarrow adder)$.
2. $(inverter, bit, word, \Gamma_{cmpl} \Rightarrow cmpl)$.

```
procedure T(G, P):
begin
  P̂ := ∅;
  J := ∅;
  k := 0;
  for each (α_1 ∧ ··· ∧ α_n → X) ∈ P do
  begin
    if X = ⊥ then
      begin
        k := k + 1
        P̂ := P̂ ∪ {α_1 ∧ ··· ∧ α_n → ⊥};
        J̄ := J̄ ∪ {(ᾱ_1, ···, ᾱ_n ⇒ false_k)};
      end
    else if X = assume(β) then
      begin
        P̂ := P̂ ∪ {goal(β) ∧ α_1 ∧ ··· ∧ α_n → assume(β)};
        J̄ := J̄ ∪ {(ᾱ_1, ···, α_n, Γ_β ⇒ β̄)};
        for j := 1 until n do P̂ := P̂ ∪ {goal(β) → goal(α_j)}
      end
    else if X = β then
      begin
        P̂ := P̂ ∪ {goal(β) ∧ α_1 ∧ ··· ∧ α_n → β};
        J̄ := J̄ ∪ {(ᾱ_1, ···, ᾱ_n ⇒ β̄)};
        for j := 1 until n do P̂ := P̂ ∪ {goal(β) → goal(α_j)}
      end
  end;
  UpdateAbstractedLabels(J̄, ADA);
  L_G := GetAbstractedLabel(Ḡ, ADA);
  for i := 1 until k do
    begin
      L_i := GetAbstractedLabel(false_i, ADA);
      for each E_G ∈ L_G do
        for each E_i ∈ L_i do
          if E_i ⊆ E_G then for (ᾱ_1, ···, ᾱ_n ⇒ false_i) ∈ J do
              for j := 1 until n do P̂ := P̂ ∪ {→ goal(α_j)}
    end;
  P̂ := P̂ ∪ {→ goal(G)};
  return P̂
end.
```

Figure 3: **Transformation Algorithm with Abstracted Dependency Analysis**

3. $(adder, cmpl \Rightarrow subtracter)$.

4. $(adder, cell \Rightarrow area\_for\_Adder)$.

5. $(cmpl, cell \Rightarrow area\_for\_Cmpl)$.

6. $(subtracter, cell, cell \Rightarrow area\_for\_Subtracter)$.

7. $(area\_for\_Adder, area\_limit \Rightarrow false_7)$.

8. $(area\_for\_Cmpl, area\_limit \Rightarrow false_8)$.

9. $(area\_for\_Subtracter, area\_limit \Rightarrow false_9)$.

10. $(\Rightarrow cla)$, $(\Rightarrow bit)$, $(\Rightarrow cell)$, $(\Rightarrow inverter)$, $(\Rightarrow word)$, $(\Rightarrow area\_limit)$.

These abstracted justifications are given to the abstracted dependency analyzer. As the result of the abstracted dependency analysis, we have the abstracted labels of those negative clauses as follows:

- $L(false_7) : \{\{adder\}\}$.
- $L(false_8) : \{\{cmpl\}\}$.
- $L(false_9) : \{\{adder, cmpl\}\}$.

Suppose that the given goal $G = adder(X)$, then the abstracted label $L(adder)$ of the proposition $adder$ is $\{\{adder\}\}$. Here, the condition of the theorem is checked for each negative clause as follows:

- The element $\{adder\}$ in $L(false_7)$ is a subset of the element $\{adder\}$ in $L(adder)$.
- No element in $L(false_8)$ is a subset of the element $\{adder\}$ in $L(adder)$.
- No element in $L(false_9)$ is a subset of the element $\{adder\}$ in $L(adder)$.

Therefore, both the negative clause (8) and the negative clause (9) are irrelevant to the goal, however, the negative clause (7) has to be evaluated for consistency checking of the solution. As a consequence, the transformed program P'' is

$$P' \setminus \{\rightarrow goal(area\_for\_Cmpl(sk1)), \rightarrow goal(area\_for\_Subtracter(sk1))\},$$

where P' is shown in Example 1. Neither the subgoal $area\_for\_Cmpl(X)$ nor the subgoal $area\_for\_Subtracter(X)$ has to be evaluated according to the abstracted dependency analysis, so that the bottom-up hypothetical reasoner derives neither the subgoal $subtracter(X)$ nor the subgoal $cmpl(X)$. Then, the computational cost to evaluate on $adder(X)$ decreases.

Another example, which is very simple, is shown in [16].

7 Experimental Result

As an example of hypothetical reasoning, we have taken up the design of logic circuits to calculate the greatest common divisor (GCD) of two integers expressed in 8 bits by using the Euclidean algorithm. The solutions are circuits calculating GCD and satisfying given constraints on area and time [13]. The program P contains several kinds of knowledge: datapath design, component design, technology mapping, CMOS standard cells and constraints on area and time [15]. In P, there are 10 normal defaults and 50 Horn clauses. The design problem of calculators for GCD includes design of components such as subtracters and adders.

Table 1 shows the expermental result, on a Pseudo-Multi PSI system, for the evaluation of the transformed programs. The run time of the original program P for a goal G is denoted by $T_{R(G,P)}$. The predicate symbol $\bar{G}$ of each goal G is *adder* (design of adders), *subtracter* (design of subtracters) or $cGCD$ (design of calculators for GCD). The run time $T_{R(G,P)}$ of each goal G is equal to the others on the original program P because bottom-up hypothetical reasoner is not goal-directed.

Let P' be the simple transformed program of P. The experiment on the transformation time shows that it takes 6.35 [s] for transforming P into P'. However, the run time $T_{R(G,P')}$ for each goal G is nearly equal to the others because constraints on area and time of the GCD calculators are represented by negative clauses. Even if we want to design adders or subtracters, the bottom up hypothetical reasoner cannot avoid designing GCD calculators for consistency checking. This is the same reason as Example 1.

Let P'' be the transformed program with the abstracted dependency analysis. The experiment on the transformation time with the abstracted dependency analysis shows that it takes 6.63 [s] for transforming P into P''. The transformation time with the abstracted dependency analysis is a little bit longer (0.28 [s]) than the previous transformation time. When $\bar{G}$ is *adder* or *subtracter*, the run time $T_{R(G,P'')}$ is much shorter than the run time for the design of GCD calculators. This is because the program can avoid consistency checks for negative clauses representing constraints on area and time of the GCD calculators when the design of adders or the design of subtracters is given as a goal. The result show that each total of the transformation time with abstracted dependency analysis and the run time of the transformed program is shorter than the run time of the original program when the problem does not need the whole of the program.

Table 1: **Run Time of Program**

Goal $\bar{G}$	$T_{R(G,P)}$ [s]	$T_{R(G,P')}$ [s]	$T_{R(G,P'')}$ [s]
adder	10.7	17.5	0.4
subtracter	10.7	17.3	0.6
$cGCD$	10.7	17.3	16.8

8 Related Work

In [14], the algorithm for first-order Horn-clause abduction with the ATMS is presented. The system is basically a consumer architecture [5] introducing backward-chaining consumers. The algorithm avoids both redundant proofs by introducing goal-directed backward-chaining consumers and duplicate proofs among different contexts by using the ATMS. Their problem definition is the same as [21], whose inputs are a goal and a set of Horn clauses. They consider a limited form of negative clauses whose antecedents are restricted to assumptions. These negative clauses are used to check the consistency of each set of assumptions and treated as forward-chaining consumers. On the other hand, since we only simulate backward-chaining by the bottom-up reasoner, we do not require both types of chaining rules. Moreover, we see that when the program includes negative clauses of free forms, it is difficult to represent the clauses as a set of consumers in general. For example, suppose that the axioms are

$$\{a \rightarrow c,\ b \rightarrow d,\ c \wedge d \rightarrow g,\ c \rightarrow e,\ d \rightarrow f,\ e \wedge f \rightarrow \bot\}$$

and the goal is g. Assume that the set of consumers is

$$\{(c \Leftarrow a),\ (d \Leftarrow b),\ (g \Leftarrow c, d),\ (e \Leftarrow c),\ (f \Leftarrow d),\ (e, f \Rightarrow \bot)\},$$

where "$\Leftarrow$" means a backward-chaining consumer and "$\Rightarrow$" means a forward-chaining consumer. Then, we get the solution $\{\langle g, \{\{g\}, \{a, b\}, \{a, d\}, \{c, b\}, \{c, d\}\}\rangle\}$. However, the correct solution is $\{\langle g, \{\{g\}\}\rangle\}$ because $\{a, b\}, \{a, d\}, \{c, b\}$ and $\{c, d\}$ are nogood. To guarantee the consistency when the program includes negative clauses, for every Horn clause, we have to add the corresponding forward chaining consumer. Such added consumers would cause the same problem as the program that appeared in using the simple transformation algorithm in Section 5.

In [22], deduction and abduction with the upside-down meta-interpretation are proposed. This abduction does not require the consistency of solutions. Furthermore, rules may do duplicate firing in different contexts since it does not use the ATMS. This often causes a problem when we apply it to practical programs where heavy procedures are attached to rules.

In contrast to our implementation, an implementation incorporating bottom-up information into top-down hypothetical reasoning has been presented in [11] for propositional programs.

Another difference between the frameworks of [14, 22, 11] and ours is that their frameworks treat only hypotheses in the form of normal defaults without prerequisites, whereas we allow for normal defaults with prerequisites.

9 Conclusion

We have presented a new transformation algorithm of programs for efficient bottom-up hypothetical reasoning based on the upside-down meta-interpretation. In the transformation algorithm, logical dependencies between a goal and negative clauses are analyzed at the abstracted level to find irrelevant negative clauses, so that consistency checking of negative clauses can be restricted to those relevant clauses. It has been evaluated with a logic circuit design problem on a Pseudo-Multi-PSI system.

It should be noted that we can apply the abstracted dependency analysis to programs transformed by Magic Set and Alexander methods, and to programs which work other bottom-up hypothetical reasoners which never do duplicate proofs. Actually, we have evaluated programs transformed by this method on several kinds of bottom-up hypothetical reasoning systems. The experimental results are shown in [10] which includes hypothetical reasoners without using the ATMS.

This method analyzes logical dependencies between assumable-predicate symbols and predicate symbols of the goal and negative clauses. Then, it maybe happens that bottom-up hypothetical reasoners perform consistency checking for some instantiated negative clauses that are irrelevant to the goal. In such a case, we could analyze dependencies on predicate symbols with their some arguments. However, this pre-analysis cost would increase much more. We conjecture that the dependency analysis at the predicate symbol level offers a reasonable way to reduce the overall effort.

Acknowledgments

Thanks are due to Mr. Makoto Nakashima of JIPDEC for implementing the ATMS and combining it with the MGTP. We are grateful to Prof. Mitsuru Ishizuka of the University of Tokyo for the helpful discussion. We would also like to thank Dr. Ryuzo Hasegawa and Mr. Miyuki Koshimura for providing us the MGTP, and Prof. Koichi Furukawa for his advise. Finally, we would like to express our appreciation to Dr. Kazuhiro Fuchi, Director of ICOT Research Center, who provided us with the opportunity to conduct this research.

References

[1] Bancilhon, F., Maier, D., Sagiv, Y. and Ullman, J.D., "Magic Sets and Other Strange Ways to Implement Logic Programs", *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pp.1–15, 1986.

[2] Beeri, C. and Ramakrishnan, R., "On the power of Magic", *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pp.269–283, 1987.

[3] Bry, F., "Query Evaluation in Recursive Databases: Bottom-Up and Top-Down Reconciled", *Data & Knowledge Engineering*, 5, pp.289–312, 1990.

[4] de Kleer, J., "An Assumption-based TMS", *Artificial Intelligence*, 28, pp.127–162, 1986.

[5] de Kleer, J., "Problem Solving with the ATMS", *Artificial Intelligence*, 28, pp.197–224, 1986.

[6] Flann, N.S., Dietterich, T.G. and Corpron, D.R., "Forward Chaining Logic Programming with the ATMS", *Proc. AAAI-87*, pp.24–29, 1987.

[7] Forgy, C.L., "Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem", *Artificial Intelligence*, 19, pp.17–37, 1982.

[8] Fujita, H. and Hasegawa, R., "A Model Generation Theorem Prover in KL1 Using a Ramified-Stack Algorithm", *Proc. 8th Int. Conf. Logic Programming*, pp.494–500, 1991.

[9] Inoue, K., "Problem Solving with Hypothetical Reasoning", *Proc. Int. Conf. on Fifth Generation Computer Systems 1988*, Tokyo, Japan, pp.1275–1281, 1988.

[10] Inoue, K., Ohta, Y., Hasegawa, R. and Nakashima.M., "Hypothetical Reasoning Systems on the MGTP", ICOT Technical Report TR-763, Institute for New Generation Computer Technology, Tokyo, Japan, 1992.

[11] Ishizuka, M. and Ito, F., "Fast Hypothetical Reasoning System using Inference-Path Network", *Proc. 3rd IEEE Int. Conf. on Tools for Artificial Intelligence*, San Jose, CA, pp.1275 1281, 1991.

[12] Junker, U., "Reasoning in Multiple Contexts", Working Paper 334, GMD, Germany, 1988.

[13] Maruyama, F., Kakuda, T., Masunaga, Y., Minoda, Y.,Sawada, S. and Kawato, N., "coLODEX: A Cooperative Expert System for Logic Design", *Proc. Int. Conf. on Fifth Generation Computer Systems 1988*, Tokyo, Japan, pp.1299–1306, 1988.

[14] Ng, H.T. and Mooney, R.J., "An Efficient First-Order Horn-Clause Abduction System Based on the ATMS", *Proc. AAAI-91*, pp.494-499, 1991.

[15] Ohta, Y. and Inoue, K., "A Forward-Chaining Multiple-Context Reasoner and Its Application to Logic Design", *Proc. 2nd IEEE Int. Conf. on Tools for Artificial Intelligence*, Heldon, VA-Nov., pp.386–392, 1990.

[16] Ohta, Y. and Inoue, K., "A Forward-Chaining Hypothetical Reasoner Based on Upside-Down Meta-Interpretation", *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, Tokyo, Japan, pp.522 529, 1992.

[17] Poole, D. , Goebel, R. and Aleliunas, R. , "Theorist: A Logical Reasoning System for Defaults and Diagnosis", N. Cercone and G. McCalla (Eds.), *The Knowledge Frontier: Essays in the Representation of Knowledge*, Springer-Verlag, pp.331–352, 1987.

[18] Poole, D. , "Compiling a Default Reasoning System into Prolog", *New Generation Computing*, 9, pp.3 38, 1991.

[19] Reiter, R. , "A Logic for Default Reasoning", *Artificial Intelligence*, 13, pp.81–132, 1980.

[20] Rohmer, J., Lescoeur, R. and Kerisit, J.M., "The Alexander Method A Technique for The Processing of Recursive Axioms in Deductive Databases", *New Generation Computing*, 4, pp.273–285, 1986.

[21] Stickel, M.E., "Rationale and Methods for Abductive Reasoning in Natural-Language Interpretation", Studer, R. (Ed.), *Proc. Int. Scientific Symp. Natural Language and Logic, Hamburg, Germany, 1989, Lecture Notes in Artificial Intelligence*, 459, Springer-Verlag, pp.233 252, 1990.

[22] Stickel, M.E., "Upside-Down Meta-Interpretation of the Model Elimination Theorem-Proving Procedure for Deduction and Abduction", ICOT Technical Report TR-664, Institute for New Generation Computer Technology, Tokyo, Japan, 1991.

[23] Ueda, K. and Chikayama, T., "Design of the Kernel Language for the Parallel Inference Machine", *The Computer Journal*, 33, 6, pp. 494–500, 1990.